CPU Scheduling Simulation and Memory Management

Andrew Alvarez, Amanda Baker, Viktorio Garev

Professor Roya Hosseini

CSC 4320/6320 - Operating Systems

GitHub Repository: <https://github.com/vgarev1/OS-Project-1>

Spring 2025

This project serves as a simulation of critical operating system functions—CPU scheduling and memory management. The main objective of the program is to demonstrate how an operating system handles multiple processes while ensuring efficient CPU utilization and memory allocation. The simulation begins by reading a structured input file containing a list of processes. Each process is described by a unique process ID (PID), its arrival time, required CPU burst time, and priority level. These inputs represent typical data an operating system receives from user programs or system tasks.  
 After reading the input, the program prompts the user to select from three CPU scheduling algorithms: First-Come, First-Served (FCFS), Shortest Job First (SJF), and Priority Scheduling. Each algorithm reflects a different approach to handling processes and illustrates how varying strategies impact system performance. During execution, the simulation calculates important performance metrics for each process, including waiting time, turnaround time, and response time. It also prints a Gantt chart to visually represent the order and duration of each process's execution.  
 An additional layer of realism is added by incorporating memory management through the First-Fit allocation strategy. In this model, each process requests a memory block based on its burst time. The First-Fit strategy searches for the first available block large enough to fulfill the request. If no block is sufficient, the process is skipped. This feature illustrates how limited memory resources can impact process scheduling and completion.

The entire project was implemented using the C programming language due to its close interaction with system-level operations. The core of the program revolves around a structured data type (struct) to encapsulate all process attributes. Each process record stores information such as the process ID, arrival and burst times, priority, start and completion times, memory block assignment, and execution state. The state can transition from READY to RUNNING to TERMINATED as the process moves through the scheduling system.  
 Memory management is designed using a fixed-size array representing predefined memory blocks with capacities of 100, 500, 200, 300, and 600 units. Each block is initially marked as available. When a process is selected for execution, the program checks the memory blocks to find the first suitable fit based on the process’s burst time. If found, the memory is allocated; if not, the process is skipped, simulating real-world memory constraints. Memory is released immediately after process completion, making it available for subsequent tasks.  
 The three scheduling algorithms are implemented as separate functions for modularity. FCFS operates by selecting processes strictly based on their arrival times, giving a straightforward and predictable order of execution. SJF improves on FCFS by choosing the process with the shortest burst time from those ready to execute, optimizing for lower average waiting and turnaround times but risking starvation for longer processes. Priority Scheduling selects processes based on their priority values, allowing higher-priority tasks to preempt lower-priority ones if necessary. In all algorithms, the system updates and tracks each process’s performance metrics, ensuring accurate calculations of total and average response, waiting, and turnaround times.

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.  
  
 One of the major challenges faced during implementation was synchronizing the memory allocation process with the scheduling algorithms. It was crucial to ensure that memory was not only allocated correctly but also freed in a timely manner to prevent deadlocks or resource starvation. Additionally, accurately calculating response times, especially when multiple processes competed for CPU time and memory, required careful handling of time variables and process states. Extensive testing and debugging were performed to validate that all performance metrics were computed correctly and that memory management behaved as expected.

In conclusion, this project effectively models the complexity of CPU scheduling and memory management within an operating system. By simulating FCFS, SJF, and Priority Scheduling algorithms alongside First-Fit memory allocation, the program provides a detailed insight into how operating systems manage concurrent processes and finite resources. Each scheduling strategy demonstrated unique strengths and weaknesses, reflecting real-world trade-offs between fairness, efficiency, and priority handling. The memory management layer reinforced the concept that resource limitations significantly influence process execution order and system throughput.  
 Through this simulation, we gained valuable experience in low-level programming, process control, memory allocation strategies, and performance evaluation. The modular design of the project ensures that future enhancements, such as adding preemptive scheduling techniques or more sophisticated memory allocation methods like Best-Fit or Worst-Fit, can be integrated with ease. Overall, the project provided a practical and comprehensive understanding of key operating system principles vital for any systems programmer or computer scientist.

Works Cited

Silberschatz, Abraham, Peter Baer Galvin, and Greg Gagne. Operating System Concepts. 10th ed., Wiley, 2018.